REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggesstions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any oenalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
	Technical Report	-

4. TITLE AND SUBTITLE
Distributed Reinforcement Learning for PolicySynchronization in Infinite-Horizon Dec-POMDPs

5a. CONTRACT NUMBER
W911NF-11-1-0124

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS
Bikramjit Banerjee, Landon Kraemer

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
University of Southern Mississippi
2609 West 4th Street
Bond Hall Room 214
Hattiesburg, MS 39406 -5157

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
57785-NS.6

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
In many multi-agent tasks, agents face uncertainty about the environment, the outcomes of their actions, and the behaviors of other agents. Dec-POMDPs offer a powerful modeling framework for sequential, cooperative, multiagent tasks under uncertainty. Solution techniques for infinite-horizon Dec-POMDPs have assumed prior knowledge of the model and have required centralized solvers. We propose

15. SUBJECT TERMS
Dec-POMDPs, reinforcement learning, multi-agent learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Bikramjit Banerjee
UU	UU	UU	UU		19b. TELEPHONE NUMBER 601-266-6287

Report Title

Distributed Reinforcement Learning for PolicySynchronization in Infinite-Horizon Dec-POMDPs

ABSTRACT

In many multi-agent tasks, agents face uncertainty about the environment, the outcomes of their actions, and the behaviors of other agents. Dec-POMDPs offer a powerful modeling framework for sequential, cooperative, multiagent tasks under uncertainty. Solution techniques for infinite-horizon Dec-POMDPs have assumed prior knowledge of the model and have required centralized solvers. We propose a method for learning Dec-POMDP solutions in a distributed fashion. We identify the issue of policy synchronization that distributed learners face and propose incorporating rewards into their learned model representations to ameliorate it. Most importantly, we show that even if rewards are not visible to agents during policy execution, exploiting the information contained in reward signals during learning is still beneficial.

Distributed Reinforcement Learning for Policy Synchronization in Infinite-Horizon Dec-POMDPs

Anonymous Author(s)
Affiliation
Address
email

Abstract

In many multi-agent tasks, agents face uncertainty about the environment, the outcomes of their actions, and the behaviors of other agents. Dec-POMDPs offer a powerful modeling framework for sequential, cooperative, multiagent tasks under uncertainty. Solution techniques for infinite-horizon Dec-POMDPs have assumed prior knowledge of the model and have required centralized solvers. We propose a method for *learning* Dec-POMDP solutions in a distributed fashion. We identify the issue of *policy synchronization* that distributed learners face and propose incorporating rewards into their learned model representations to ameliorate it. Most importantly, we show that even if rewards are not visible to agents during policy execution, exploiting the information contained in reward signals during learning is still beneficial.

1 Introduction

In many multi-agent tasks, agents face uncertainty about the environment, the outcomes of their actions, and the behaviors of other agents. Dec-POMDPs offer a powerful modeling framework for sequential, cooperative, multiagent tasks under uncertainty.

State-of-the art infinite-horizon Dec-POMDP solution techniques such as Pajarinen and Petonen's PERI [1] and Amato and Zilberstein's Goal–directed approach [2] rely on knowledge of model parameters and centralized solvers; however, in some scenarios such knowledge may not be available and using centralized solvers may be unfeasible. The vast majority of Dec-POMDP solution techniques (for both finite and infinite horizons) have either relied on such prior knowledge of the environment or have utilized centralized solvers.

Recently, some algorithms have been proposed for distributed learning policies for finite-horizon Dec-POMDPs. Zhang and Lesser [3] proposed a scalable distributed, model-free method for a special class of Dec-POMDP where agents are organized in a network, and Banerjee and Kraemer [4] proposed a distributed, model-free method for the general (non-networked) case of finite-horizon Dec-POMDPs; however, both of these methods rely on an learning a Q-value function that grows exponentially in the horizon length. Thus, such methods cannot be applied to learning in infinite-horizon Dec-POMDPs.

Learning algorithms for infinite-horizon POMDPs (which are essentally single-agent Dec-POMDPs) have long existed. In this work, we leverage the Utile Distinction Memory algorithm (UDM) [5], which was developed to solve infinite-horizon POMDPs, to solve infinite-horizon Dec-POMDPs. UDM is a model-based learning algorithm, in that a UDM agent attempts to *learn* its own POMDP model and find the best policy for that model.

Our work with this algorithm (which we call "UDM-Alt"), has revealed an interesting issue that we believe can arise not only in UDM-Alt but other such distributed algorithms for learning infinite Dec-

POMDP policies. We discuss the difficulty distributed infinite-horizon learners have with what we call *policy synchronization* in Section 6.1, and discuss how reward information can be incorporated to ameliorate this problem.

The rewards agents receive from the environment can carry a significant amount of information. To our knowledge, previously Dec-POMDP policy representations have not exploited this; however, the argument can be made that rewards are more of a conceptual feature, existing to define the problem and provide an evaluation criteria for policies. However, rewards must be made available while the agents are learning otherwise the policy they learn is meaningless. With this in mind, we create a variant of UDM-Alt that only requires rewards during the learning phase (i.e. it requires no rewards during policy execution), and we show that this algorithm is capable of finding better policies that UDM-Alt (which incorporates no reward information).

2 Decentralized POMDPs

The Decentralized POMDP (Dec-POMDP) formalism extends the POMDP formalism to accommodate multiple agents. We can define a Dec-POMDP as a tuple $\langle n, S, A, P, R, \Omega, O \rangle$, where:

- n is the number of agents playing the game.
- S is a finite set of (unobservable) environment states.
- $A = \times_i A_i$ is a set of joint actions, where A_i is the set of individual actions that agent i can perform.
- $P(s'|s, \vec{a})$ gives the probability of transitioning to state $s' \in S$ when joint action $\vec{a} \in A$ is taken in state $s \in S$.
- $R(s, \vec{a})$ gives the immediate reward the agents receive upon executing action $\vec{a} \in A$ in state $s \in S$.
- $\Omega = \times_i \Omega_i$ is the set of joint observations, where Ω_i is the finite set of individual observations that agent i can receive from the environment.
- $P(\vec{\omega}|s', \vec{a})$ gives the probability of the agents jointly observing $\vec{\omega} \in \Omega$ if the current state is $s' \in S$ and the previous joint action was $\vec{a} \in A$.

In Dec-POMDPs, it is generally assumed that agents cannot communicate their observations and actions to each other. These constraints are often present in real world scenarios, where communication may be expensive or unreliable. Consider, for instance, a scenario in which a team of robots must coordinate to search a disaster area for survivors. In such a task, robots may need to spread out to efficiently cover the area and also may need to travel deep underneath rubble, both of which could interfere with wireless communication.

Assuming agents cannot communicate observations and actions, each agent must choose actions based only upon his own actions and observations; however, since the transition, reward, and observation functions depend on *joint* actions, the quality of each agent's policy is dependent on the policies played by all agents or the *joint policy*. The goal of the Dec-POMDP problem, then, is to find the joint policy that maximizes the expected reward received by the agents. The problem of finding this optimal joint policy has been proven to be NEXP-complete [6].

3 Alternating Best-Response for Dec-POMDPs

In [7], Nair et. al present the JESP algorithm for solving finite-horizon Dec-POMDPs in which a centralized solver, with complete knowledge of the Dec-POMDP model, finds a joint policy by alternately *computing* each agent's best response policy to the policies of the other agents. More recently, Kraemer and Banerjee [4] proposed Q-Alt, an alternating method for solving finite-horizon Dec-POMDPs that requires no knowledge of the underlying Dec-POMDP model.

Algorithm 1 gives an abstract definition of these alternating best-response solution methods. π represents the current joint policy, with the policy for a given agent i denoted as π_i. The variable $finished$ represents some termination condition. In JESP, $finished$ is true when π stops changing, while in Q-Alt $finished$ is true after a set number of alternations. The function $findBestResponse(i, \pi)$

Algorithm 1 SOLVEALTERNATELY

1: $\pi \leftarrow initializePolicies()$
2: **while** not $finished$ **do**
3: **for** $i = 1$ to n **do**
4: $\pi_i \leftarrow findBestResponse(i, \pi)$
5: **end for**
6: **end while**

returns agent i's best response policy to the current policies of the other agents, which is *computed* in JESP but *learned* in Q-Alt.

In Q-Alt, $findBestResponse(i, \pi)$ involves learning Q-Values [8] of the form $Q(h_t, a|\pi \setminus \pi_i)$ where h_t is an individual action-observation history of length t and $a \in A_i$. It is not assumed, however, that agent i knows $\pi \setminus \pi_i$, so these Q-Values can be written as $Q(h_t, a)$. Essentially, agent i is treating the other agents as part of the environment as it is learning a policy for a finite-horizon POMDP (i.e. a single-agent Dec-POMDP) with stochastic rewards. Unfortunately, for a given agent i, there can be $O(|A_i|^t|\Omega_i|^t)$ individual action-observation histories, and, therefore, learning these Q-Values is intractable for infinite-horizon Dec-POMDPs.

Thus, in order to use this alternate learning approach for an infinite-horizon Dec-POMDP, we require a method for learning policies for infinite-horizon POMDPs. In this work, we propose a new algorithm based upon 1 in which agents use McCallum's UDM algorithm [5] for learning policies (i.e. UDM is used for $findBestResponse(i, \pi)$). We refer to the resulting algorithm as UDM-Alt.

4 Utile Distinction Memory for Dec-POMDPs

In [9], Chrisman proposed a method for learning POMDP policies in which a learning agent constructs its own model and then uses replicated Q-Learning to find a policy. The agent's local model is essentially a POMDP model, having a set of *local* states S, a transition model over the classes $P(s'|s, a)$, and an observation model $P(\omega|s', a)$, where $s, s' \in S$ are local states, $\omega \in \Omega$ is an observation, and $a \in A$ is an action. Note that local states are not necessarily the same as the underlying Dec-POMDP model states. A given local state s may be a probability distribution over the true model states as well as the set of policies the other agent could be following. Also, note that in Chrisman's work, the observation distribution was based only upon the state (i.e. the action was irrelevant); however, in order to address problems with information-gathering actions such as DecTiger, we include the previous action in the observation model.

As the agent interacts with the environment, it maintains a *belief* distribution $b_t \in \Delta S$, where $b_t(s)$ indicates the agent's belief that it is in local state s at step t. During policy execution where the agent only has the series of actions it has executed and observations it has seen up to a given step t, the agent's belief is given by the *forward* probability α from the Baum-Welch forward-backward procedure [10].

$$\begin{aligned} \alpha_t(s') = \quad & P(\omega_2, \ldots, \omega_t, s_t = s'|a_1, \ldots, a_{t-1}) \\ & = z \cdot P(\omega_t|s', a_{t-1}) \textstyle\sum_{s \in S} \alpha_{t-1}(s) P(s'|s, a_{t-1}) \end{aligned} \tag{1}$$

where z is a normalizing factor. However, once an agent has executed up to T steps for $T > t$ it is able to improve $b_t(s)$ by considering how well its experience after t is explained by $s_t = s$. This probability is known as the *backward* probability β_t in Baum-Welch and is given by

$$\begin{aligned} \beta_t(s) = \quad & P(\omega_{t+1}, \ldots, \ldots \omega_T|s_t = s, a_{t+1}, \ldots, a_{T-1}) \\ & = z \cdot \textstyle\sum_{s' \in S} \beta_{t+1}(s') P(\omega_{t+1}|s', a_t) P(s'|s, a_t). \end{aligned} \tag{2}$$

Along with the model parameters, the agent also maintains a Q-Value function $Q : S \times A \mapsto \mathbb{R}$, where $Q(s, a)$ gives an estimation of the expected reward attained by executing action a in state s. At a time step t, the agent selects which action to execute according to $\arg\max_{a \in A} \sum_{s \in S} b_t(s) Q(s, a)$.

Since, the agent has only a probability distribution over states, it cannot always identify a single $Q(s,a)$ to update at a given step as is possible in Q-Learning for MDPs. Instead, it must update $Q(s,a)$ for all $s \in S$ since it may have had a small probability of being in any state. After executing an action at step t and receiving reward r, the agent maintains its Q-Values using

$$\forall s \in S, Q(s,a_t) = Q(s,a_t) + \delta b_t(s) \left[r + \gamma \max_{a'} \sum_{s' \in S} b_{t+1}(s') Q(s',a') \right] \tag{3}$$

where δ is the learning rate, and γ is a discount factor.

Chrisman's algorithm consists of a series of trials with each trial having two main stages. In the first stage, the agent interacts with the environment for T steps, updating its Q-Values as it goes along. In the second main stage of a trial, the agent updates its internal model. In this second stage, the agent first updates its model parameters using the Baum-Welch algorithm. For each step t in the history, there is an associated action a_t, observation ω_{t+1}, and reward r_{t+1}. At the end of a trial, an agent updates its model using the following. Let

$$\xi_t(s,s') = z_{\xi_t} \cdot \alpha_t(s) P(s'|s,a_t) P(\omega_{t+1}|s',a_t) \beta_{t+1}(s') \tag{4}$$

where z_{ξ_t} is a normalizing constant that ensures $\sum_{s,s'} \xi_t(s,s') = 1$.

$$\forall a \in A, \forall s,s' \in S, P(s'|s,a) = z_P(s,a) \cdot \sum_{t=1}^{T} I(a_t = a) \xi_t(s,s') \tag{5}$$

$$\forall a \in A, \forall \omega \in \Omega, \forall s' \in S, P(\omega|s',a) = z_O(s',a) \sum_{t=1}^{T} I(a_t = a, \omega_{t+1} = \omega) \sum_{s} \xi_t(s,s') \tag{6}$$

where $I(event)$ returns 1 if $event$ occurred and 0 if it did not occur and where $z_P(s,a)$ and $z_O(s',a)$ are normalizing constants which ensure that $\sum_{s' \in S} P(s'|s,a) = 1$ and $\sum_{\omega \in \Omega} P(\omega|s',a) = 1$, respectively. Baum-Welch will continue to iteratively update the model parameters until the model converges.

The agent's local states may alias many of the states of the true underlying model, and it may be the case that the agent needs to execute different actions in those different underlying states. To address this, after Baum-Welch has completed the agent attempts to make new *perceptual distinctions*. While Chrisman's perceptual distinction criterion was based solely upon expected frequencies in model of transitions and observations, in [5], McCallum instead considered the utility of making a perceptual distinction as the criterion. This method is known as the "Utile Distinction Memory" (UDM) approach, and it differs from Chrisman's only in the perceptual distinction criterion.

5 Informed Policy

Note that step 1 of Algorithm 1 requires that π be initialized. Since the agents learn in an alternating fashion, all agents that have not yet learned a policy will need an initial policy to play. As noted in [4] initializing these policies in an *informed* manner can produce better results. The informed policy presented used in those works is incompatible with infinite-horizon Dec-POMDPs, so we present a way to repurpose UDM for learning an informed initial policy for infinite-horizon Dec-POMDPs. We make the assumption that agents can share their exploration mechanisms beforehand, as was done in [4], and therefore, while each agent maintains its own model while learning the initial policy, every agent learns the same model. Rather than alternate learning, the agents update and follow their policies concurrently.

First, each agent creates a UDM model with $S = s_1, s_2, \ldots s_k$ where $k = |A|$, i.e. one state for each possible *joint* action. Each agent also creates a state $s_\emptyset$ which is unreachable from every state, i.e. $\forall a \in A, \forall s \in S, P(s_\emptyset|s,a) = 0$. b_0 is initialized such that $s_\emptyset$ is always the start state. Let s_a denote the state associated with action $a \in A$. For all joint actions $a \in A$ and for all states $s,s' \in S$, if $s' = s_a$, then $P(s'|s,a) = 1$, otherwise $P(s'|s,a) = 0$. In other words, the only way to reach state s_a is via *joint* action a. Conceptually, each state in the model, including those that may be added by UDM, represents a joint action-only history; however, an action-only history of length t', $h_{t'}$ can still bring our agents to a state s which represents h_t, $t \geq t'$ so long as the last t' steps of h'_t match

the last t' steps of h_t. The benefit in using UDM as opposed to the method proposed in [4] is that we do not have to learn directly a Q-Value for each action-only history h_t and action a (which would be intractable for large t) but rather we can learn a Q-Values only for portions of history that are significant.

6 Exploiting Rewards

6.1 Addressing Desynchronization

In this section, we discuss a particular miscoordination issue that can arise in execution of infinite-horizon Dec-POMDP policies using an example from the Dec-Tiger domain defined in [7]. In the Dec-Tiger domain, there are two doors. Behind one door is treasure, but behind the other door is a tiger. Opening the door concealing the treasure yields a reward, and opening the door concealing the tiger yields a penalty. The two agents do not know a priori which door the tiger is behind; however, they are allowed to listen for clues. Upon listening, each agent either hears the tiger behind the left door or the right door. The observations are noisy, however, and each agent has a fifteen percent chance of hearing the tiger behind the wrong door. Miscoordination is heavily penalized. If an agent opens a door and the other agent does not open the same door any reward received will be halved, and likewise any penalty received will be doubled.

Figure 1 depicts the execution of a joint policy that we observed in our experiments with UDM-Alt. In this policy, an agent opens a door D if it has received at least two observations since the last time it opened a door (or since it began an episode), and the number of "Hear Not D" observations it has observed is greater than the "Hear D" observations it has observed. In this policy, when an agent opens a door, it (erroneously) assumes that the other agent has also opened the door. This is similar to known optimal finite-horizon DecTiger policies; however, in those policies, an agent will only decide to open a door at specific steps (this is a key distinction which we will revisit shortly). In Figure 1, the rows correspond to execution steps. The columns labeled t and Tiger report the step number and location of the tiger, respectively. For a given step t, a_i is the action agent i executes at that step and b_i represents the agent's belief *before* executing that action. The column labeled ω_i reports the observation agent i received after executing the action in the previous step.

We can see from the table that both agents listen for the first two steps. After the first two steps, agent 2 has heard the tiger twice behind the left door and is therefore 97% certain that the tiger is behind the left door, so it chooses to open the right door. Agent 1, however, received conflicting observations and therefore has no idea which door the tiger is behind. At step 4, Agent 1 is unaware that that Agent 2 has opened a door and therefore it does not know that the observation received carries no information, nor does it know that the state has been reset. Agent 2 knows both of these things, and adjusts its belief state accordingly. After this point, the agents become unsynchronized, and without being able to discern when the other agent has opened a door, it does not seem likely that they can become synchronized again. As mentioned previously, the optimal finite-horizon policies for DecTiger avoid this issue because agents only open doors at the last step for horizons 3-5 and at the third and last steps for horizon 6.

These synchronization points seem to be the key for solving infinite-horizon DecTiger, and in fact, PERI [1], the algorithm which produced the policy with the highest known value (13.45) for infinite-horizon DecTiger exploited this. PERI assumes that tasks are periodic, and therefore can easily create a controller that resets every X steps as is required in DecTiger. Indeed, we can show formally that simply looping the optimal horizon-3 DecTiger policy has the same expected value as was achieved with PERI. The value for the horizon-3 policy when the discount factor is 0.9 is the discounted sum of the expected value at each step $v = -2 \cdot (0.9)^0 + -2 \cdot (0.9)^1 + 9.190812 \cdot (0.9)^2 = 3.6439$. Playing this periodic policy repeatedly will yield an expected value of $V = v(0.9^0)^0 + v(0.9^3)^1 + v(0.9^3)^2 + \ldots$. Since this is an infinite geometric series, we can find the sum using $\frac{v}{1-.9^3} = 13.45$.

6.2 Incorporating Reward Information

As mentioned previously, if an agent could observe when its counterpart has opened a door, this desynchronization issue could be avoided. Unfortunately, it is generally assumed that agents are

unable to observe the actions performed by other agents. In fact, in alternating learning, a learning agent, having no knowledge of the behavior of the other agent, treats the other agents as part of the environment. This means that a given $P(s'|s,a)$ or $P(\omega|s',a)$ an agent has estimated is actually marginalized over the possible actions of the other agents A_- as follows.

$$P(s'|s,a) = \sum_{a_- \in A_-} P(s'|s,a,a_-)P(a_-|s) \tag{7}$$

$$P(\omega|s',a) = \sum_{a_- \in A_-} P(\omega|a,a_-)P(a_-|s) \tag{8}$$

So, this ambiguity can affect not only policy execution as in Figure 1 but also model and policy learning.

Now, UDM (and consequently our alternating UDM algorithm) relies on the fact that agent(s) receive a reward upon executing an action (or joint action). Depending upon the domain, a given reward r can carry a significant amount of information.

In the DecTiger problem, for instance, agents receive a reward of -2 when they both listen, and various other rewards if at least one of them opens a door. Therefore, if an agent listens and receives a reward that is not -2, it can be certain that the other agent opened a door.

Along with disambiguating the other agent's actions, reward can also carry information about the previous state. For instance, whenever agents open the correct door in DecTiger, they receive a positive reward of either 9 if only one agent opened the door while the other listened, or 20 if both agents opened the door. So, if an agent performs the action "Open D" for some door D and receives either 9 or 20, it can know with certainty that the previous state was "Not D". Of course, since the state resets upon opening a door, this is of little value for DecTiger policy execution; however, it could greatly improve the accuracy of the $\xi(s,s')$ (equation 4) used to update the agent's internal model. That is, rewards can help the agent learn a better model faster. Furthermore, in other domains it may be the case that knowing a state with certainty could be beneficial during execution as well.

An agent can incorporate reward information into its UDM model by conditioning the transition and observation probabilities on r (i.e. $P(s'|s,a,r)$ and $P(\omega|s',a,r)$) and by maintaining a new estimator $P(r|s,a)$. The resulting changes to the model leads us to the following modified representations of α and β (originally defined in equations 1 and 2).

$$\begin{aligned}\alpha'_t(s') = & \quad P(\omega_1, r_1 \ldots, \omega_t, r_t, s_t = s'|a_1, \ldots, a_{t-1}) \\ & = z_{\alpha'_t} \cdot P(\omega_t|s', a_{t-1}, r_t) \textstyle\sum_{s \in S} \alpha'_{t-1}(s) P(s'|s, a_{t-1}, r_t) P(r_t|s, a_{t-1})\end{aligned} \tag{9}$$

$$\begin{aligned}\beta'_t(s) = & \quad P(\omega_{t+1}, r_{t+1}, \ldots, \ldots \omega_T, r_T|s_t = s, a_{t+1}, \ldots, a_{T-1}) \\ & = z_{\beta'_t} \cdot P(r_{t+1}|s, a_t) \textstyle\sum_{s' \in S} \beta'_{t+1}(s') P(\omega_{t+1}|s', a_t) P(s'|s, a_t, r_{t+1})\end{aligned} \tag{10}$$

Using this new representation of α and β, the agent can proceed through trials as before, updating $P(s'|s,a,r)$ and $P(\omega|s',a,r)$ in a manner similar to 5 and 6 with the modified indicators $I(a_t = a, r = r_{t+1})$ and $I(a_t = a, w_{t+1} = w, r_{t+1} = r)$, respectively. The estimator $P(r|s',a)$ is updated using

$$P(r|s,a) = z \cdot \sum_{t=1}^{T} \sum_{s' \in S} I(a_t = a, r = r_{t+1}) \xi'_t(s,s'). \tag{11}$$

We refer to UDM-Alt with using these modifications as UDM-Alt-R$^+$.

It should be noted that agents using UDM-Alt-R$^+$ will still treat other agents as part of the environment. In fact, $P(s'|s,a,r)$ and $P(\omega|s',a,r)$ are still marginalized over A_- as follows.

$$P(s'|s,a,r) = \sum_{a_- \in A_-} P(s'|s,a,r,a_-)P(r|s,a,a_-)P(a_-|s,a) \tag{12}$$

$$P(\omega|s',a,r) = \sum_{a_- \in A_-} P(\omega|a,r,a_-)P(r|s,a,a_-)P(a_-|s,a) \tag{13}$$

However, the extra reward information allows an agent to essentially marginalize over subsets of A_{-} instead of the entire set because $P(r|s, a, a_{-})$ can be zero if a certain reward is impossible given the states in the underlying model represented by local state s and a, and a_{-}.

Figure 2 gives an example of a policy discovered by UDM-Alt-R^{+}. Each node is labeled by an action, and the transition edges are labeled with one or more vectors having the form $\langle reward, observation \rangle$. An asterisk in the place of a reward or observation, denotes that any symbol can match that element, and $\overline{-2}$ represents all rewards other than -2. The policy is executed as follows. The agent executes the action in the top-most node, and then it receives a reward and observation. It then follows the transition from that node that matches the reward-observation it has received. Note that this policy is *not* periodic. That is, agents will not automatically reset their policy execution after X number of steps. Instead, the agents only reset their policy execution when they receive a reward indicating that a door was open. The periodic policy mentioned previously essentially forces the agents to discard everything they have learned about the location of the tiger after three steps. In the policy we present in Figure 2, the agents only discard their belief about the tiger when they are sure it is no longer valid. Thus, this policy has a slight advantage and achieves a value of 14.11 (greater than the previously established 13.45), but this is not entirely surprising considering that this policy assumes that extra information is available (i.e. the reward).

t	Tiger	ω_1	ω_2	a_1	a_2	b_1	b_2
1	Left	-	-	L	L	<.50, .50>	<.50, .50>
2	Left	HL	HL	L	L	<.85, .15>	<.85, .15>
3	Left	HR	HL	L	OR	<.50, .50>	<.97, .03>
4	Left	HL	HL	L	L	**<.85, .15>**	**<.50, .50>**
5	Left	HL	HL	OR	L	<.97, .03>	<.85, .15>
6	Right	HL	HL	L	OR	<.50, .50>	<.97, .03>

Figure 1: An example of agents becoming unsynchronized. Each row represents a step of execution, a_i represents the action executed at that step, ω_i represents the observation the agent i received after executing a_i in the previous step, and b_i represents agent i's belief prior to executing a_i.

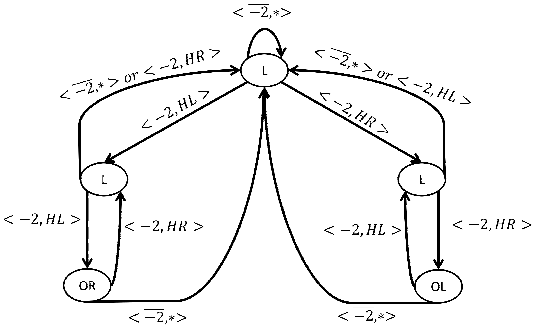

Figure 2: A policy extracted from UDM-Alt for DecTiger that incorporates rewards. Nodes are labeled with actions and transition edges are labeled with reward-observation vectors $< r, \omega >$. If the reward-observation vector received by an agent in a given node, matches the transition labeling an edge transitioning from that node, the agent follows that transition.

6.3 Removing Reliance on Rewards

In some scenarios, it may not be reasonable to assume that reward information is available during policy execution. Indeed, reward may only exist as a concept to define the problem and to provide an evaluation criteria for policies. If this is the case, the UDM-Alt-R^{+} will not be usable. However, UDM-Alt requires rewards during the learning phase, and therefore if we can use UDM-Alt for learning then we have access to rewards during the learning phase. To this end, we have developed a variant of UDM-Alt that uses rewards during the learning phase but does not require rewards during policy execution. We accomplish this by simply marginalizing over the rewards when propagating beliefs during policy execution. In other words, we use UDM-Alt-R^{+} during model learning, but we generate our beliefs during policy execution using

$$b_t(s) = \alpha''_t(s') = z \cdot \sum_r P(\omega_t|s', a_{t-1}, r) \sum_{s \in S} \alpha''_{t-1}(s) P(s'|s, a_{t-1}, r) P(r|s, a_{t-1})$$

In essence, when an agent marginalizes over rewards, it is making a guess about the reward it received based upon prior experience. Unfortunately, such predictions will not give the agent much insight into the current state; however, it may allow it to model the behaviors of the other agents more effectively.

We refer to this variant of UDM-Alt which marginalizes over rewards as UDM-Alt-R$^-$.

7 Experiments

We evaluated our three UDM-Alt variants (UDM-Alt, UDM-Alt-R$^+$, and UDM-Alt-R$^-$) for five different benchmark problems. For all runs, we initialized π as per Section 5 using 20 trials having 50 episodes each with each episode consisting of 200 steps. For each benchmark problem, we ran each variant of UDM-Alt 50 times (with a different random seed each time). For each run, we allowed the agents to learn for two alternations (i.e. each agent had two turns learning) with each alternation consisting of 50 trials each having 50 episodes each with each episode having 500 steps. The learning rate δ was .005, and we used epsilon-greedy exploration [8] with $\epsilon = .2$. This discount factor γ was .9 for all benchmark problems. For each run, we evaluated the quality of the resulting policy by simulating for 100000 episodes of 150 steps and averaging the reward attained during those episodes.

Our results are reported in table 1. For each benchmark problem, we report the best known policy value (which were reported in [11, 1], and the method which attained it. For each of our UDM-Alt variants, we report the solution quality averaged over 50 runs, and, in parentheses, the error relative to the best known value ϵ_r.

Problem $(\|S\|, \|A_i\|, \|\Omega_i\|)$	Best Known (Method)	UDM-Alt-R$^+$ (ϵ_r)	UDM-Alt-R$^-$ (ϵ_r)	UDM-Alt (ϵ_r)
BroadcastChannel $(2, 2, 2)$	9.10 (NLP)	9.10 (0)	9.10 (0)	9.10 (0)
DecTiger $(2, 3, 2)$	13.45 (PERI)	13.24 (0.02)	-5.77 (1.43)	-18.843 (2.4)
Recycling $(4, 3, 2)$	31.93 (Mealy NLP)	21.90 (.31)	21.91 (0.31)	21.53 (0.33)
2x2 Grid $(16, 5, 2)$	6.89 (PERI)	3.59 (.49)	3.60 (0.48)	3.80 (.45)
Box Pushing $(100, 4, 5)$	149.85 (Goal-Dir.)	18.01 (0.88)	8.58 (0.94)	-5.03 (1.03)

Table 1: Results Table

From the table we can see that all variants of UDM-Alt perform exceptionally well on the BroadcastChannel problem. This is not surprising, however, because BroadcastChannel is relatively easier than the other problems. Incorporating rewards was obviously helpful in DecTiger as the relative error is only .02 for UDM-Alt-R$^+$. While UDM-Alt-R$^-$ does not fare as well for DecTiger, it clearly performs better than UDM-Alt. This suggests that incorporating rewards into the learning process can be beneficial, even if rewards are not available during policy execution. All three variants perform reasonably well on the Recycling Robots; however, incorporating reward appears to have no effect on the result. Similarly, in the Meeting on a 2x2 Grid problem, reward appears to have no effect on the result.

All three variants of UDM-Alt show remarkably poor performance on the Box Pushing problem (though the two variants incorporating reward perform slightly better). Such poor performance is likely due to the fact that, while agents can coordinate to move large box together for a larger reward, they have the option of pushing smaller boxes individually for a smaller reward, which can lead to a local optima. We think resolving this issue may be a good avenue for future work.

8 Conclusion

We have proposed three algorithms for solving infinite-horizon Dec-POMDPs in a distributed manner without knowledge of the model, and we have evaluated the quality of the solutions they produce on five benchmark problems. We have discussed the danger that distributed learners face of becoming unsynchronized during Dec-POMDP policy execution, and have discussed how incorporating reward information into our learning algorithm can help agents synchronize policies and can improve the quality of the learned model. We have shown empirically that incorporating rewards into our models during the learning process can improve the resulting values of policies learned, even if those policies are not allowed to incorporate rewards.

References

[1] Joni Pajarinen and Jaakko Peltonen. Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2636–2644, December 2011.

[2] Christopher Amato and Shlomo Zilberstein. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 593–600, Budapest, Hungary, 2009.

[3] C. Zhang and V. Lesser. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Proc. AAAI-11*, San Francisco, CA, 2011.

[4] Landon Kraemer and Bikramjit Banerjee. Informed initial policies for learning in dec-pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence Student Abstract and Poster Program*, Toronto, Canada, July 2012. To appear.

[5] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995.

[6] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[7] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, Acapulco, Mexico, 2003.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[9] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Articial Intelligence*, pages 183–188, San Jose, CA, 1992. AAAI Press.

[10] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[11] C. Amato, D.S. Bernstein, and S. Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proc. UAI*, 2007.